2.7.1 — Foundations of Proof Systems

Exam

2017-2018

1 Warming up...

Question 1 Give a proof in natural deduction of the following proposition :

$$(f \implies (g \implies h)) \implies ((f \implies g) \implies (f \implies h)).$$ ◇

Solution.

$$\dfrac{\dfrac{\dfrac{f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash f \quad f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash f \Rightarrow (g \Rightarrow h)}{f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash g \Rightarrow h} \quad \dfrac{f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash f \quad f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash f \Rightarrow g}{f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash g}}{f \Rightarrow (g \Rightarrow h); f \Rightarrow g; f \vdash h}}{f \Rightarrow (g \Rightarrow h); f \Rightarrow g \vdash f \Rightarrow h}}{f \Rightarrow (g \Rightarrow h) \vdash (f \Rightarrow g) \Rightarrow (f \Rightarrow h)}}{(f \Rightarrow (g \Rightarrow h)) \Rightarrow ((f \Rightarrow g) \Rightarrow (f \Rightarrow h))}$$

Question 2 Consider a closed term t in Martin-Löf type theory whose type is :

$$t : \forall\, n : \mathrm{nat}\,.\ \Sigma\, p : \mathrm{nat}\,.\ \mathrm{prime}\ p \wedge (n < 3 \vee (n < p \wedge (\forall\, k : \mathrm{nat}\,.\, n < k < p \to \neg\, \mathrm{prime}\, k))).$$

where prime p is a predicate that holds iff p is a prime number.

Give the normal form of $\pi_1(t\ 8)$ where π_1 is the first projection for Σ-types. ◇

Solution. The normal form is 11 (the smallest prime number larger than 8). □

Question 3 Give a closed term whose type is :

$$\forall (A\ B : \mathsf{Type})(P : A \to B \to \mathsf{Prop}).$$
$$(\Sigma(y : B).\ \forall (x : A).\, P\ x\ y) \to (\forall (x : A).\ \Sigma(y : B).\, P\ x\ y).$$ ◇

Solution. $\lambda A : \mathsf{Type}.\lambda B : \mathsf{Type}.\lambda P : A \to B \to \mathsf{Prop}.\lambda c : \Sigma(y : B).\ \forall (x : A).\, P\ x\ y.\lambda x : A.(\pi_1(c), \pi_2(c)\ x)$. □

2 Strong vs Weak Induction

Question 4 How would you prove in type theory the following judgement :

$$\begin{aligned}&\forall(P : \text{nat} \to \mathsf{Prop}).\\ &\quad(\forall(n : \text{nat}).\,(\forall(p : \text{nat}).\,p < n \to P\ p) \to P\ n) \to\\ &\quad\forall(n : \text{nat}).\,P\ n\end{aligned}$$

from the following induction principle :

$$\begin{aligned}&\forall(P : \text{nat} \to \mathsf{Prop}).\\ &\quad P\ 0 \to\\ &\quad(\forall(n : \text{nat}).\,P\ n{-}{>}\ P\ (\text{S}\ n)) \to\\ &\quad\forall(n : \text{nat}).\,P\ n\end{aligned}$$

Solution. One has to use the induction scheme with the following predicate over n : $\forall m, m \leq n \to P\ m$.

This predicate is true for 0 (because $(P\ 0)$).

If it is true for n, then we have in particular $(P\ n)$, so $(P\ (S\ n))$ also holds and the predicate is true for all $m \leq (S\ n)$.

3 Limited Principle of Omniscience

We define the following three propositions.

$$\text{EM} \triangleq \forall(P : \mathsf{Prop}).\,P \vee \neg P$$

$$\text{LPO} \triangleq \forall(P : \text{nat} \to \text{bool}).\,(\forall(n : \text{nat}).\,P\ n = \bot) \vee (\Sigma(n : \text{nat}).\,P\ n = \top)$$

$$\begin{aligned}\text{LLPO} \triangleq\ &\forall(P : \text{nat} \to \text{bool}).\\ &(\forall(n\ p : \text{nat}).\,P\ n = \top \wedge P\ p = \top \to n = p) \to\\ &(\forall(n : \text{nat}).\,P(2i) = \bot) \vee (\forall(n : \text{nat}),\,P(2i+1) = \bot)\end{aligned}$$

Question 5 Prove constructively that EM → LPO and that LPO → LLPO. ◇

We now give a variant of LPO where the predicates in consideration are not necessarily decidable :

$$\text{LPPO} \triangleq \forall(P : \text{nat} \to \mathsf{Prop}).\,(\forall(n : \text{nat}).\,\neg P\ n) \vee (\Sigma(n : \text{nat}).\,P\ n)$$

Question 6 Prove that LPPO → EM. ◇

Solution. [EM → LPO] — let P : nat → bool. By EM, we know that $\Sigma(n : \text{nat}).\, P\ n = \top$ or $\neg(\Sigma(n : \text{nat}).\, P\ n = \top)$. In the first case, we are done. Otherwise, we prove that $\forall(n : \text{nat}).\, P\ n = \bot$. Assume that $h : \neg(\Sigma(n : \text{nat}).\, P\ n = \top)$ and let n_0 : nat. Since $P\ n_0$: bool, we can do a case analysis on $P\ n_0$. If $P\ n_0 = \bot$, we are done. If $P\ n_0 = \top$, then $\Sigma(n : \text{nat}).\, P\ n = \top$. We can conclude using h.

[LPO → LLPO] — let P : nat → bool and assume that

$$h : \forall(n\ p : nat).\, P\ n = \top \wedge P\ p = \top \to n = p.$$

Let Qe $i := P\ (2i)$ and Qo $i := P\ (2i + 1)$. By LPO on Qe, we know that $\forall(n : \text{nat}).\, P\ (2i) = \bot$ or $\Sigma(n : \text{nat}).\, P\ (2i) = \top$. In the first case, we are done. Otherwise, by LPO on Qo, we know that $\forall(n : \text{nat}).\, P\ (2i + 1) = \bot$ or $\Sigma(n : \text{nat}).\, P\ (2i + 1) = \top$. Yet again, in the first case we are done. It remains the case where we know two natural numbers n_e and n_o s.t. $P\ (2n_e) = \top$ and $P\ (2n_o + 1) = \top$. By h, we have that $2n_e = 2n_o + 1$, obtaining a contradiction. □

4 Being even

We remind the usual definition of addition in Coq :

```
Fixpoint add n m :=
 match n with
  | 0 => m
  | S p => S (add p m)
end.
```

Question 7 Recall *very briefly* how one proves :

```
Lemma add_nS : forall n m, n+(S m) = S n + m.
```

◇

Solution. By induction over n, using the computational behavior of `add` in both cases. □

The following is a possible definition of the property of being even in Coq :

```
Inductive even : nat -> Prop :=
| E0 : even 0
| ESS : forall n, even n -> even (S (S n)).
```

Question 8 What is the elimination scheme associated to this definition ? ◇

Solution.
```
even_ind
     : forall P : nat -> Prop,
       P 0 ->
       (forall n : nat, even n -> P n -> P (S (S n))) ->
       forall n : nat, even n -> P n
```

A friend comes up with the following alternative definition :

```
Definition evs (n : nat) : Prop :=
  exists x, n = x + x.
```

Question 9 We want to show that `forall n, even n -> evs n`. What would the main steps be ? ◇

Solution. By induction over the proof of `(even n)`. Two cases :

— $n = 0$, in which case we have to prove `exists x, 0 = x+x` which is done by providing 0 as the witness.

— In the second case we have to prove `exists x, (S (S n)) = x+x` knowing `exists x, n = x+x`. We eliminate the latter hypothesis to obtain `x` and provide `(S x)` as the witness. The proof boils down to `(S (S n)) =(S x)+(S x)` which is proved using `add_Sn`. □

Question 10 Conversely, how would you prove `forall n, evs n -> even n`. ◇

Solution. One proves `even (n+n)` by induction over `n`. The result follows easily. □

5 Recursive and inductive predicates

We consider the usual definition of natural numbers in Coq (with constructors `O` and `S`) and the following (usual) definition of lists :

```
Inductive list : Type :=
  nil : list
| cons : nat -> list -> list.
```

Question 11 Here are four predicates defined by case analysis and recursion. For each of them, give an equivalent inductive predicate of the same type. You do not have to give the proof that your formulation is equivalent to the given predicate. However, *try to give an inductive predicate that is as concise and elegant as possible*. Every time, give also the type of the generated elimination principle.

```
Fixpoint N1 (n : nat) : Prop :=
  match n with
  | O => True
  | S O => False
  | S (S O) => False
  | S (S (S m)) => N1 m
  end.

Fixpoint L1 (l : list) : Prop :=
  match l with
  | nil => True
  | cons n l' => (n1 n)/\(L1 l')
  end.

Fixpoint L2 (n:nat)(l:list) : Prop :=
```

```
  match l with
  | nil => False
  | cons m l' => n=m \/ L2 n l'
  end.


Fixpoint addChain (l : list) : Prop :=
  match l with
  | nil => False
  | cons (S O) nil => True
  | cons _ nil => False
  | cons m l' => exists x, exists y, (L2 x l')/\(L2 y l')/\m=x+y/\addChain l'
  end.
```

Solution.
```
Inductive N1p : nat -> Prop :=
  NI0 : N1p 0
| N1S : forall n, N1p n -> N1p (S (S (S n))).

N1p_ind
     : forall P : nat -> Prop,
       P 0 ->
       (forall n : nat, N1p n -> P n -> P (S (S (S n)))) ->
       forall n : nat, N1p n -> P n

Inductive L1p : list -> Prop :=
| L1p0 : L1p nil
| L1pc : forall n l, N1 n -> L1p l -> L1p (cons n l).

L1p_ind
     : forall P : list -> Prop,
       P nil ->
       (forall (n : nat) (l : list), N1 n -> L1p l -> P l -> P (cons n l)) ->
       forall l : list, L1p l -> P l



Inductive L2p : nat -> list -> Prop :=
| L2p1 : forall l n m, L2p n l -> L2p n (cons m l)
| L2p2 : forall l n, L2p n (cons n l).

L2p_ind
     : forall P : nat -> list -> Prop,
       (forall (l : list) (n m : nat), L2p n l -> P n l -> P n (cons m l)) ->
       (forall (l : list) (n : nat), P n (cons n l)) ->
       forall (n : nat) (l : list), L2p n l -> P n l
```

```
(* alternative definition *)
Inductive L2pp (n : nat) : list -> Prop :=
| L2pp1 : forall l m, L2pp n l -> L2pp n (cons m l)
| L2pp2 : forall l, L2pp n (cons n l).


Inductive ADC : list -> Prop :=
| ADC1 : ADC (cons 1 nil)
| ADCc : forall l a b, L2 a l -> L2 b l -> ADC l -> ADC (cons (a+b) l).

ADC_ind
     : forall P : list -> Prop,
       P (cons 1 nil) ->
       (forall (l : list) (a b : nat),
        L2 a l -> L2 b l -> ADC l -> P l -> P (cons (a + b) l)) ->
       forall l : list, ADC l -> P l
```

Question 12 (optional) The last predicate characterizes so called *addition chains*. An addition chain ending with number n gives a way to compute a^n in a time proportional to the length of the list. Can you see how ? ◇

Finding the shortest addition chain ending with n is an open problem. There is no known reasonably efficient algorithm.

Solution. Once one has computed a^n and a^m one can compute a^{n+m} with one single multiplication. So given an addition chain $[1; n_1; n_2; \ldots; n_k]$ one can compute all exponentiations a^{n_i} in k multiplications. □